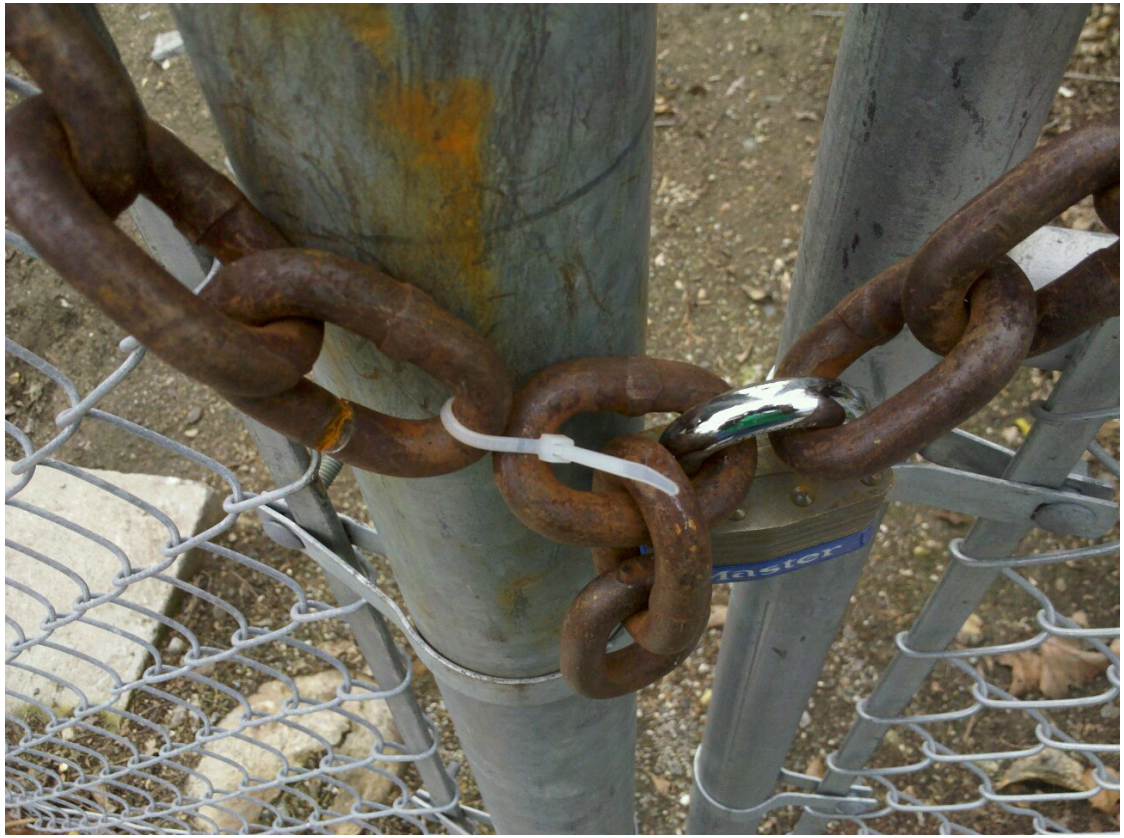
STRATEGIES FOR DEVELOPING SECURE SOFTWARE

Very Old Truism

- Chain is only as strong as its weakest link



KISS

- Make a very short chain

- Simplicity is paramount in system design
  - Makes it possible to analyse system
  - Reduces likelihood of errors

Defence in Depth

- Have parallel chains
- Use redundancy when implementing security measures
- Very simple example: fortress
  - Moat
  - Outer wall
  - inner wall

Defence in Depth (2)

- Example: network security
  - Physical Security
  - Authentication and password security
  - Antivirus software
  - Firewalls (hardware or software)
  - DMZ (Demilitarized zones)
  - IDS (Intrusion Detection Systems)
  - Proxy servers
  - VPN (Virtual private networks)
  - Logging and Auditing

Defence in Depth (3)

- However, there are some issues
- Interactions between security measures
- Side effects of security measures
  - Example: several security measures can be used for denial of service

Stratification

- Don't keep your eggs in the same basket
- Do not bundle targets for attack
  - Makes the attack more attractive/profitable
  - Example: NATO forbids transporting confidential documents and money together
- When attacker has compromised one security measure, he should have access to one asset

Stratification (2)

- Make compartments
  - Two processes that communicate with simple protocol
  - Two computers
  - Send logs to separate computer

Manage Assumptions

- Common security failure: assumption made by developers is not met
- These assumptions are often implicit and/or unknown
  - Cryptographic key is transferred securely
  - User's workstation does not contain malware
  - User is not malicious
  - OS or middleware enforces access control

Manage Assumptions (2)

- Installation or management does not correspond to assumptions
  - File permissions are not set up properly
  - Administrator does not perform periodical checks
- Environment for system changes
  - System installed on different platform
  - System for internal use is opened up to partners and customers

Manage assumptions (3)

- Always write down explicitly what your security solution assumes
- Always find out what your tools and components really provide
  - Smart card provides secure place for storing keys and secure environment for calculating signatures
  - It does not ensure that correct document is signed
  - Find out which assumptions the components make

Manage Assumptions (4)

- Avoid the Titanic effect: relying on some state-of-the-art security measure

Manage Change

- Periodically inspect the assumption list to see whether they still hold
- When implementing new features:
  - Check that they do not introduce vulnerabilities
  - Especially when creating exceptions to general case
  - Beware of feature interaction

Trustno1

- Do not trust processes that do not run under your control
  - Client software
  - Browser
  - Partners
- Never rely on client-side integrity checks
- Do not store state in other computers

Trustno1 (2)

- Do not trust yourself
- Attacker will introduce faults
  - Physically tampering
  - Tampering with environment
  - Tampering with memory
- Software will contain errors
- Check integrity of state and fail, if state is not consistent

Trusted Computing Base

- TCB is part of the system that is able to enforce your security policy
- Be explicit about your TCB
  - Does it include OS, hardware, etc?
- Keep the TCB small

Security by Obscurity

- Kerckhoff's principle: a cryptosystem should be secure even if everything about the system, except the key, is public knowledge
- Do not rely on the design of your system being secure
- It **will** be reverse-engineered (and possibly published)

Security by Obscurity (2)

- Examples of reverse engineered systems
  - A5/1 cipher (GSM)
  - RC4
  - CSS (DVD encryption)
  - SMB (Windows file sharing)
- In crypto, security by obscurity is almost always bad
  - You will not get peer review before implementation
  - You **will** get reviews after it is in production

Security by Obscurity (3)

- It can be used as additional security measure
  - Should not be relied on
  - Think of it as „delaying attack” or „increasing costs of attack”
- Assume that it will be broken
- Useful, if you are smart
  - Example: military can use good cryptographers

Security by Obscurity (4)

- Can hinder competition
  - Microsoft protocols and file formats
  - Skype protocol
    - P. Biondi and F. Desclaux, "Silver Needle in the Skype"
- Frequent changes are a good idea

Open vs. Closed Source

- Linus's law: „given enough eyeballs, all bugs are shallow”

However, …

- There is no strong evidence
- Open source software can also be ridden with security holes
- Security reviewing is boring and usually not very rewarding task
- Open source developers are not inherently better than closed source developers
- Closed source developers can have strong security-oriented process

Security by Diversity

- Monocultures can be vulnerable
  - Single vulnerability can be used to compromise many hosts
  - Example: Windows + Internet Explorer
- Having different platforms or using minority platform can be useful
  - Less exploits developed
  - Some protection against script kiddies
  - However, this is usually more expensive

Consider People

- People are the biggest security risk
- Make security measures user friendly
  - People want to get things done
- Weave security measures into working process
  - Accounting is a good example
- Motivate people
  - Provide rewards for discovering irregularities
  - Do not punish for following security policy

Using Cryptography

- **Never, ever invent a cryptographic algorithm!**
- You will not get it right
- It is easy to create a system that you cannot break
- Industry-strength algorithms have been scrutinized by the whole world for years (decades)

Using Cryptography (2)

- Use only the most popular algorithms
  - They have received the most attention
- Symmetric crypto: 3DES, AES
- Asymmetric crypto: RSA
- Digital signature: RSA, DSA
- Hashing: SHA1, SHA256, SHA512

Using Cryptography (3)

- Do not implement algorithms
- There are lot of small things that can go wrong
  - Using bad random number generator
  - Not clearing key memory
  - Allowing keys to be swapped out
  - Using insecure keys or parameters
  - Using invalid initial values or padding
- Use popular cryptographic libraries
  - OpenSSL, CryptoAPI

Using Cryptography (4)

- Use libraries correctly
- Read the manual
  - What are the assumptions?
  - What you must provide?
  - What the library provides?
  - What each parameter and return value means?
- Understand how the library works

Secure Defaults

- System should be secure by default
  - In Windows XP, user is administrator by default
  - In Debian, administrator account is disabled by default
- Deny access by default
- Usually people will not make effort to secure it
  - Often they do not know they have to

Fail Gracefully

- On failure go to secure state
- Do not try to use heuristics and recover
  - Instead, shut down the process
- In physical security this is sometimes difficult
  - Cannot lock doors in the event of fire
- Be aware of denial of service attack

Pay Attention to Boundaries

- Attacks happen on boundary between trusted and untrusted zones
- Know when you are on a boundary
- Validate all untrusted data
  - User input
  - Environment
  - Configuration files
  - Command line parameters
  - File names